Stamina: Stabilisation Monoids IN Automata theory *

Nathanaël Fijalkow[1], Hugo Gimbert[2], Edon Kelmendi[3], and Denis Kuperberg[4]

[1] University of Warwick, United Kingdom
[2] LaBRI, Bordeaux, France
[3] TU Munich, Germany
[4] CNRS, ÉNS Lyon, France

Abstract. We present Stamina, a tool solving three algorithmic problems in automata theory. First, compute the star height of a regular language, i.e. the minimal number of nested Kleene stars needed for expressing the language with a complement-free regular expression. Second, decide limitedness for regular cost functions. Third, decide whether a probabilistic leaktight automaton has value 1, i.e. whether a probabilistic leaktight automaton accepts words with probability arbitrarily close to 1.
All three problems reduce to the computation of the stabilisation monoid associated with an automaton, which is computationally challenging because the monoid is exponentially larger than the automaton. The compact data structures used in Stamina, together with optimisations and heuristics, allow us to handle automata with several hundreds of states. This radically improves upon the performances of ACME, a similar tool solving a subset of these problems.
The tool Stamina is open source and available from Github, details are given on the webpage http://stamina.labri.fr.

1 Introduction

Stamina is a tool for deciding properties of automata, through the construction of an algebraic structure called stabilisation monoid. It solves three problems:

- compute the star height of a regular language,
- decide limitedness for regular cost functions,
- decide whether a probabilistic leaktight automaton has value 1.

The star height problem, introduced by Eggan in 1963 [Egg63], takes as input a regular language L and an integer h and decides whether there exists a regular expression for L with at most h nested Kleene stars. The minimal h having this property is called the star height of L. An excellent introduction to the star height problem is given in [Kir05], which mentions some of the important industrial applications such as speech recognition [Moh97], database theory [GT01], and image compression [IK93,KMT04].

* This work was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1. H. Gimbert and E. Kelmendi are supported by the French ANR project "Stoch-MC" and "LaBEX CPU" of Université de Bordeaux. This work has been partly funded by the grant Palse Impulsion.

This problem was considered as one of the most difficult problems in the theory of recognizable languages and it took 25 years before being solved by Hashiguchi [Has88]. Implementing Hashiguchi's algorithm is hopeless: even for a language L given by an automaton with 4 states, a "very low minorant" of the number of languages to be tested with L for equality is $c^{(c^c)}$ with $c = 10^{10^{10}}$ [LS02].

It took another 22 years before an algorithm with a better algorithmic complexity was given by Kirsten in [Kir05]. Kirsten's algorithm takes as input an automaton recognising a language L and an integer h and constructs an automaton with counters (nowadays called a B-automaton) inducing a function $f : A^* \to \mathbb{N} \cup \{\infty\}$ with the following property: f is limited if, and only if, the star height of L is at most h. Kirsten's solution was later adapted to trees [CL08a] using the framework of regular cost functions.

Stamina aims at solving the star height problem for practical applications, albeit the doubly exponential space complexity of Kirsten's algorithm is a challenge to tackle. To our best knowledge, this is the first time a solution to the star height problem is implemented.

The limitedness problem for regular cost functions takes as input a B-automaton inducing a function $f : A^* \to \mathbb{N} \cup \{\infty\}$, and checks whether the function f is bounded on its domain (words with a finite value). The theory of regular cost functions has been introduced by Colcombet [Col09,Col13], as a general formalism to express limitedness problems. A number of problems have been solved thanks to this theory (see e.g. [CL08a,CL08b,CKLB13]), and Stamina includes a general-purpose cost functions library.

The value 1 problem takes as input a probabilistic automaton and checks whether there are words accepted with probability arbitrarily close to 1. Probabilistic automata are a versatile tool widely used in speech recognition as well as a modelling tool for the control of systems with partial observations. They extend classical automata with probabilistic transitions, see [Rab63] for an introduction, and [FGO12] for the value 1 problem. This problem is a reformulation of a natural controller synthesis problem: assume a blackbox finite state system with random events is controlled by a blind controller who inputs actions to the blackbox but has absolutely no feedback on the state of the system. Then the synthesis of controllers with arbitraily high reliability is equivalent to solving the value 1 problem.

Stabilisation monoids are the key mathematical object behind the solutions to those three problems. For both B-automata and probabilistic automata, one can associate a stabilisation monoid generalising the notion of transition monoid. This monoid carries precise information about the behaviour of the automaton.

A seminal paper by Simon [Sim94] provides a combinatorial tool called the forest factorization theorem, at the heart of the solution of the limiteness problem for stabilisation monoids associated to B-automata. These algebraic techniques were adapted to solve the value 1 problem for probabilistic leaktight automata [FGO12,FGKO15].

Related work. Stamina is written in C++ and improves over a previous tool called Acme [FK14] implemented in OCaml, which was a first proof-of-concept tool using stabilisation monoids as an algorithmic back-end to solve the limitedness problem for

regular cost functions. We provide quantitative experiments showing that Stamina performs much better than Acme, thanks to several optimisations. This improvement allows us to provide a new functionality: solving the star height problem, which was unrealistic with Acme as it could not handle large automata.

2 Computing the Stabilisation Monoid

The core computation performed by Stamina is the construction of the stabilisation monoid generated by a finite set of matrices.

2.1 Stabilisation Monoids in a Nutshell

Stabilisation monoids are sets of square matrices of fixed dimension n over a finite semiring $(\mathbb{S}, +, \cdot, 0, 1)$. When solving problems related to probabilistic automata, $\mathbb{S}$ is the boolean semiring $(\{0,1\}, \vee, \wedge, 0, 1)$. When solving problems related to B-automata, including the star height problem, $\mathbb{S}$ is the semiring of sets of counter actions, see subsection 4.1.

The set of square matrices of dimension n over $\mathbb{S}$ inherits from $\mathbb{S}$ a monoid structure, where the product of two matrices is defined as usual:

$$(M \cdot N)[i,j] = \sum_{k=1}^{n} M[i,k] \cdot N[k,j] \ .$$

To obtain a stabilisation monoid one furthermore defines a unary operation on matrices called the *stabilisation* and denoted $\sharp$, which satisfies some axioms:

$$(M^\sharp)^\sharp = M^\sharp \tag{1}$$

$$(MN)^\sharp M = M(NM)^\sharp \tag{2}$$

The intuition is that every matrix M is the abstraction of a matrix $\mathbf{M}$ with coefficients in an infinite monoid (e.g. for probabilistic automata, the reals with addition and multiplication) and $M^\sharp$ represents the asymptotic behaviour of the sequence $(\mathbf{M}^n)_{n\in\mathbb{N}}$. Some more details are provided in Section 4. The formal definition of a stabilisation monoid involves an order as well [Col09], which plays no role in Stamina.

2.2 Efficient Computation of Stabilisation Monoids

We report on our implementation of the following algorithmic task:

> Given a set of matrices S, compute the smallest set of matrices containing S and stable under product and stabilisation, called the stabilisation monoid generated by S.

Since the semiring $\mathbb{S}$ is finite then the set of $n \times n$ matrices on $\mathbb{S}$ is finite as well and the stabilisation monoid generated by S is computable as follows:

Repeat
    Add to S every product $M \cdot N$ for $M, N \in S$
    Add to S every $M^\sharp$ for $M \in S$
Until no new elements are added to S

Stamina implements this naïve algorithm with two main optimisations, one saves space and the other saves time.

Saving space: unique identifiers for matrices and vectors. The generated monoid can be exponential in the size of the matrices, so the crucial aspect here is space optimisation.

An $n \times n$ matrix is not represented as a list of n^2 coefficients but as a list of $2n$ pointers to vectors representing the rows and columns of the matrix: The vectors themselves are stored in a compact way, for example on a 64 bit architecture a single integer is used to store up to 64 coefficients of the boolean semiring.

To save even more space, all vectors and matrices are stored uniquely in global hashmaps. This induces a constant time comparison for matrices and vectors, as they are equal if, and only if, their pointers are equal. This allows Stamina to handle monoids with several billions of elements, and in practice Stamina computes monoids with several millions of elements with a small memory footprint.

Saving time: rewrite rules. In our application, the initial set of matrices is given by matrices M_a for $a \in A$, where A is the finite alphabet of the automaton. Hence we naturally associate to every element of the stabilisation monoid a $\sharp$-expression, which is a term on two operations: product and stabilisation. For instance $(M_a \cdot M_b^\sharp \cdot M_a)^\sharp$ is associated to $(ab^\sharp a)^\sharp$. There are infinitely many $\sharp$-expressions and finitely many matrices thus most $\sharp$-expressions rewrite in an equivalent and shorter way. Along with the computation of the vectors and matrices of the monoid, Stamina stores a list of rewrite rules of $\sharp$-expressions to a set of minimal non-equivalent expressions.

These rewrite rules are used in conjunction with the axioms (1) and (2) in order to minimise the number of iterations of the algorithm. For example, if $MN = M^\sharp$ then $(MN)^\sharp = M^\sharp$ according to (1), so once the rewrite rule $MN \to M^\sharp$ is known, Stamina avoids the computation of $(MN)^\sharp$.

Stamina implement the inner loop of the naïve algorithm as follows. It alternates between closure by product and closure by stabilisation. In both cases, Stamina keeps a pending list of candidates for new elements. The computation of the generated monoid is over when the list of candidates is empty. For each candidate, Stamina checks whether it can be simplified by rewrite rules and axioms and in this case the candidate is dropped. Otherwise Stamina computes the corresponding matrix and checks whether this matrix is already known. If yes, Stamina creates a new rewrite rule. If not, Stamina adds a new element to the monoid.

3 Benchmarks

We compared the running times of Stamina and its predecessor Acme [FK14].

For the benchmarks we draw random automata which produce larger stabilisation monoids in order to observe the difference in performances between the two versions. The point of comparison is the size of the computed monoid rather than the number of states in the automaton, since some large automata can produce small monoids, and vice-versa, some small automata can produce large monoids.

To obtain random automata we proceed as follows. First, for each state s we pick a state t with uniform probability on the set of states and add a transition between s and t, ensuring that each state has an outgoing transition for each letter. After this we pick a number $p \in [0, 1]$ at random, and for all other states t' different from t, we add a transition between s and t' with probability p.

The results have been plotted in Figure 1. One can observe that there is a threshold in the size of the Markov monoid after Acme will not be useful, *i.e.* either takes too much time or has a stack overflow. This threshold is depicted by the vertical line in the graph below (it hovers around 3500 elements).

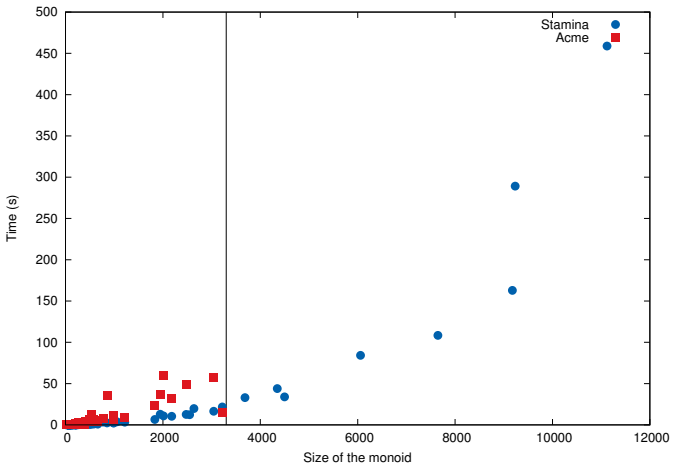


Fig. 1: Random automata of size 10.

4 Stabilisation Monoids for B- and Probabilistic Automata

The notion of stabilisation monoids appears in two distinct contexts. It has first been developed in the theory of regular cost functions, introduced by Colcombet [Col09,Col13]. The underlying ideas have then been transferred to the setting of probabilistic automata [FGO12].

4.1 Stabilisation Monoids in the Theory of Regular Cost Functions

At the heart of the theory of regular cost functions lies the equivalence between different formalisms: a logical formalism, cost MSO, two automata model, B- and S-automata, and an algebraic counterpart, stabilisation monoids.

Here we briefly describe the model of B-automata, and their transformations to stabilisation monoids. This automaton model generalises non-deterministic automata by adding a finite set of counters; instead of accepting or rejecting a word, a B-automaton associates an integer value to each input word. Formally, a B-automaton is a tuple $\mathcal{A} = \langle A, Q, \Gamma, I, F, \Delta \rangle$, where A is a finite alphabet, Q is a finite set of states, Γ is a finite set of counters, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times A \times \{\mathbf{r}, \mathbf{e}, \mathbf{ic}\}^\Gamma \times Q$ is the set of transitions. A transition (p, a, τ, q) allows the automaton to go from state p to state q while reading letter a and performing action $\tau(\gamma)$ on counter γ. The action $\mathbf{ic}$ increments the current counter value by 1, $\mathbf{e}$ leaves the counter unchanged, and $\mathbf{r}$ resets the counter to 0.

The value of a run is the maximal value assumed by any of the counters during the run. The semantics of a B-automaton $\mathcal{A}$ is defined on a word w by

$$[\![\mathcal{A}]\!](w) = \inf\{\mathrm{val}(\rho) \mid \rho \text{ is a run of } \mathcal{A} \text{ on } w\} \ .$$

In other words, the automaton uses the non determinism to minimise the value among all runs. In particular, if $\mathcal{A}$ has no run on w, then $[\![\mathcal{A}]\!](w) = \infty$.

The main decision problem in the theory of regular cost functions is the limitedness problem. We say that a B-automaton $\mathcal{A}$ is *limited* if there exists N such that for all words w, if $[\![\mathcal{A}]\!](w) < \infty$, then $[\![\mathcal{A}]\!](w) < N$.

One way to determine whether a B-automaton $\mathcal{A}$ is limited is by computing its stabilisation monoid. It contains matrices over the semiring of sets of counter actions $\{\mathbf{r}, \mathbf{e}, \mathbf{ic}, \omega\}^\Gamma$; more precisely it is the stabilisation monoid generated by the matrices corresponding to each letter. Defining the semiring of sets of counter actions is a bit tedious (see [Col09,Col13]); for the sake of explanations we will restrict ourselves to the case of one counter. As we will explain for the star height problem it is enough to work with a subclass of B-automata called hierarchical, for which this semiring also considerably simplifies.

Assuming the B-automaton $\mathcal{A}$ has only one counter, its stabilisation monoid is a set of matrices over the semiring of counter actions $\{\mathbf{r}, \mathbf{e}, \mathbf{ic}, \omega\}$ defined as follows: the addition of the semiring is the minimum for the order $\mathbf{r} < \mathbf{e} < \mathbf{ic} < \omega$, and the multiplication is the maximum for the order $\mathbf{e} \prec \mathbf{ic} \prec \mathbf{r} \prec \omega$. This semiring structure induces a product operation on matrices. See [Col09] for a formal definition of the stabilisation operation on these matrices.

We now give some intuitions about the stabilisation monoid of $\mathcal{A}$. Consider a $\sharp$-expression e, as for instance $a(ba)^\sharp$. It induces a sequence of words, in the example $(a(ba)^n)_{n \in \mathbb{N}}$. The goal is to associate to every $\sharp$-expression e a matrix M_e such that M_e summarises the action of $\mathcal{A}$ on the sequence of words induced by e. More precisely, $M_e[i, j]$ is a counter action describing the runs from i to j on the sequence of words. To illustrate, assume that for each word $a(ba)^n$ there are two runs from i to j, performing

on the counter the actions $\mathbf{e}(\mathbf{ic}\ \mathbf{e})^n$ and $\mathbf{r}(\mathbf{ic}\ \mathbf{r})^n$, respectively. The first run gives rise to $\max(\mathbf{e}, (\max(\mathbf{ic}, \mathbf{e}))^\sharp) = \max(\mathbf{e}, \omega) = \omega$, and the second to $\max(\mathbf{r}, (\max(\mathbf{ic}, \mathbf{r})^\sharp) = \max(\mathbf{r}, \mathbf{r}) = \mathbf{r}$. The summary of these two runs is $\min(\omega, \mathbf{r}) = \mathbf{r}$. The use of $\min$ and $\max$ matches the definition of a value of a word as the infimum over all runs of the maximum values of the counter.

An unlimited witness is a $\sharp$-expression inducing a sequence of words $(u_n)_{n\in\mathbb{N}}$ such that $\lim_n [\![\mathcal{A}]\!](u_n) = \infty$. As shown in [Col09,Col13], the stabilisation monoid of a B-automaton $\mathcal{A}$ contains an unlimited witness if, and only if, it is not limited. This gives a conceptually simple solution to the limitedness problem: compute the stabilisation monoid and check for the existence of unlimited witnesses.

We briefly discuss the case of hierarchical actions, as it is used for the solution to the star height problem, and correspond to the the nested distance automata in [Kir05]. We have $k+1$ counters numbered from 0 to k. The hierarchical actions are the following, for $j \in [0, k]$:

- R_j resets all counters p with $p \geq j$, the others remain unchanged;
- I_j increments the counter j, resets the counters p with $p > j$, the others remain unchanged;
- $\mathbf{e}$ leaves all counters unchanged;
- ω means that some counter reached very high values.

The addition of the semiring is the minimum for the order $R_0 < R_1 < \cdots < R_k < \mathbf{e} < I_0 < I_1 < \cdots < I_k < \omega$. The multiplication of the semiring is the maximum for the order $\mathbf{e} \prec I_k \prec R_k \prec I_{k-1} \prec R_{k-1} \prec \cdots \prec I_0 \prec R_0 \prec \omega$.

4.2 Stabilisation Monoids for Probabilistic Automata

The notion of stabilisation monoids also appeared for probabilistic automata, for the Markov Monoid Algorithm. This algorithm was introduced in [FGO12] to partially solve the value 1 problem: given a probabilistic automaton $\mathcal{A}$, does there exist $(u_n)_{n\in\mathbb{N}}$ a sequence of words such that $\lim_n \mathbb{P}_{\mathcal{A}}(u_n) = 1$?

Although the value 1 problem is undecidable, it has been shown that the Markov Monoid Algorithm correctly determines whether a probabilistic automaton has value 1 under the *leaktight* restriction. It has been recently shown that all classes of probabilistic automata for which the value 1 problem has been shown decidable are included in the class of leaktight automata [FGKO15], hence the Markov Monoid Algorithm is the *most correct* algorithm known to (partially) solve the value 1 problem.

As for the case of B-automata, the stabilisation monoid of a probabilistic automaton is the stabilisation monoid generated by the set of matrices corresponding to each letter. The underlying semiring is the Boolean semiring; the definition of the stabilisation is specific to probabilistic automata, we refer to [FGO12,FGKO15] for details.

5 The Star Height algorithm

The latest algorithm in the literature for computing star height is designed for tree automata [CL08a], but we will use it here in the special case of words. The main improve-

ment over the previous algorithm from [Kir05] is the identification of the structure of Subset Automata, which allows minimisation. We discuss the main ideas of the algorithm.

5.1 Subset Automata

We consider deterministic automata with ϵ-transitions, *i.e.* such that the transition relation is of the form $\Delta \subseteq Q \times (A \cup \{\epsilon\}) \times Q$. One can see the ϵ-transitions as defining a partial order on states.

Definition 1 ([CL08a]). *A subset automaton $\mathcal{A}$ is a deterministic automaton with ϵ-transitions such that:*

- *The ϵ-transitions induce a sup-semi-lattice, i.e. every subset P of Q has a least upper bound $\bigvee P$. In particular, there is a minimum element $\bigvee \emptyset$ and a maximal element $\bigvee Q$.*
- *The transition function of $\mathcal{A}$ is compatible with the sup-semi-lattice structure, i.e. for all $P \subseteq Q$ and $a \in A$, we have $\delta(\bigvee P, a) = \bigvee\{\delta(p, a) \mid p \in P\}$.*

It is proved in [CL08a] that any regular language can be recognized by a subset automaton, which can be obtained by a powerset construction from a non-deterministic automaton for the complement language. Note however that this subset automaton is of exponential size in the original automaton.

An interesting property of subset automata is that they can be minimised. The states of the minimal subset automaton are intersection of residuals of the language [CL08a]. We implemented the minimisation algorithm, which turns out to be a precious optimisation.

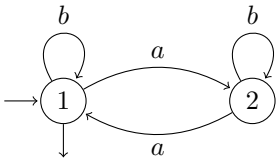
5.2 Reduction to the Limitedness Problem for B-Automata

We start from a subset automaton recognising the language L and an integer k, and want to determine whether L has star height at most k. We construct a hierarchical B-automaton $\mathcal{B}$ with $k+1$ counters such that L has star height at most k if, and only if, $\mathcal{B}$ is limited.

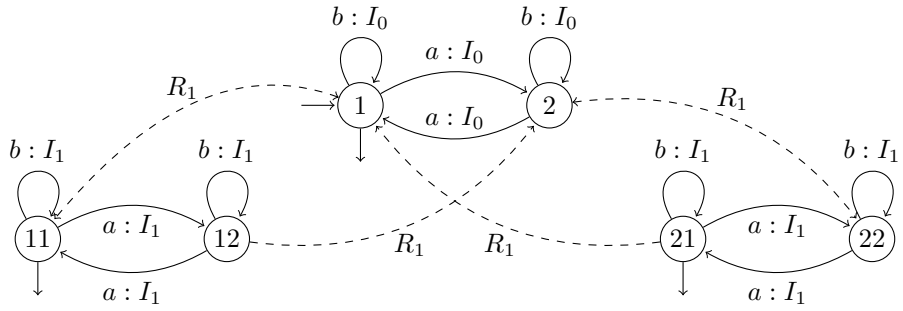
The set of states is $Q' = \bigcup_{i=1}^{k+1} Q^i$, which we view as a subset of Q^*. The initial state is q_0. A state $\rho \cdot p$ is final if, and only if, $p \in F$. We now define the transitions:

- If $(p, a, q) \in \Delta$ and $\rho \in Q^{\leq k}$, we add the transition $(\rho \cdot p, a, I_{|\rho|}, \rho \cdot q)$. If $a = \varepsilon$, the action may equivalently be replaced by e, as we do in the implementation.
- If $\rho \in Q^{\leq k-1}$ and $p \in Q$, we add the transition $(\rho \cdot p, \epsilon, R_{|\rho \cdot p|}, \rho \cdot p \cdot p)$.
- If $\rho \in Q^{\leq k-1}$ and $p, q \in Q$, we add the transition $(\rho \cdot p \cdot q, \epsilon, R_{|\rho \cdot p|}, \rho \cdot q)$.

Example 1. We apply the construction to the following automaton $\mathcal{A}$, and want to determine whether it has star height at most 1. The minimal subset automaton of $\mathcal{A}$ happens to be isomorphic to $\mathcal{A}$ in this case.



We construct the following B-automaton $\mathcal{B}$, where the ε-transitions are the dashed transitions.



Therefore, for any fixed k, we can decide in EXPSPACE whether a regular language given by a deterministic automaton has star height at most k. The algorithm is the following:

- first construct a subset automaton recognising the same language by a powerset construction, yielding an exponentially bigger automaton,
- minimise the subset automaton,
- construct the B-automaton $\mathcal{B}$ as above,
- check $\mathcal{B}$ for limitedness using the stabilisation monoid algorithm, which means constructing the stabilisation monoid of $\mathcal{B}$, of exponential size in $\mathcal{B}$.

To obtain an EXPSPACE upper bound one needs to perform the computation of the stabilisation monoid on the fly using polynomial space. Note that Kirsten's algorithm uses doubly exponential space because it takes a non-deterministic automaton as input.

The algorithmic task that we want to perform is the following: given a regular language L, compute its star height. The section above describes how, given a language L and an integer k, we can test whether L has star height at most k, by constructing a B-automaton $\mathcal{B}(k)$ such that L has star height at most k if, and only if, $\mathcal{B}(k)$ is limited.

One may thus simply apply the algorithm above for increasing values of k:

$k = 0$
Repeat
    Construct $\mathcal{B}(k)$
    If $\mathcal{B}(k)$ is not limited, increment k

Until $\mathcal{B}(k)$ is limited
Return k

The automaton $\mathcal{B}(k)$ becomes rather quickly very large. This means that checking it for limitedness may be intractable. We know that there are short witnesses that $\mathcal{B}(k)$ is not limited, given by unlimited witnesses. We will show in the next section how the loop complexity heuristic provides us with potential such witnesses.

5.3 The Loop Complexity Heuristic

We present a decisive optimisation, based on the notion of loop complexity introduced by Eggan in his seminal paper [Egg63]. Although it is only a heuristic, it led to huge improvements in test cases.

The *loop complexity* of an automaton $\mathcal{A}$, denoted $\mathrm{LC}(\mathcal{A})$, is the star height of an expression obtained from the automaton via a standard algorithm. It has been extensively studied, and many properties of the loop complexity are known. For instance, the star height of a language L is the loop complexity of *some* automaton recognising L [Egg63,LS02]. There are many natural cases where the star height is equal to the loop complexity, for instance when all transition labels are distinct, see [Coh70] for further results in this direction.

Computing the loop complexity is very efficient and can be carried out in polynomial time. Denote e_{LC} the regular expression witnessing the loop complexity. We use this expression for two purposes:

- First, it provides an upper bound for the star height.
- Second, the regular expression e_{LC} induces a list of $\sharp$-expressions that are potential unlimitedness witnesses in $\mathcal{B}(k)$.

The point is that computing both e_{LC} and the potential witnesses is very fast compared to actually checking whether $\mathcal{B}(k)$ is limited. Hence this gives fast means to observe that $\mathcal{B}(k)$ is unlimited without having to compute its stabilisation monoid.

Starting from a regular expression, we construct a list of $\sharp$-expressions in the following way.

- First, we say that a regular expression is in normal form if the sums appear only at the root or directly below Kleene stars, *i.e.* not under a product. One can easily rewrite a regular expression into one in normal form by distributing sums over products.
- Given a regular expression in normal form, one obtains a $\sharp$-expression by inductively transforming $(\sum_i e_i)^*$ into $(\prod_i e_i^\sharp)^\sharp$. For instance, $(a+b)^*$ becomes $(a^\sharp b^\sharp)^\sharp$. The idea behind this translation if to make the most of loops in the automaton.
- A regular expression e in normal form is a sum of regular expressions e_i, each in normal form. The list of $\sharp$-expressions for e is obtained by applying the previous transformation to each e_i.

Let us return to the example presented above, Example 1. Its loop complexity is 2, and the expression computed by the algorithm is $e_{\mathrm{LC}} = (b + (ab^*a))^*$. This regular expression is turned into the $\sharp$-expression $(b^\sharp(ab^\sharp a)^\sharp)^\sharp$ by our heuristic. It turns out that this $\sharp$-expression is an unlimited witness for $\mathcal{B}(1)$. The simplest such witness is $(b^\sharp ab^\sharp a)^\sharp$. This shows that in this example the Loop Complexity heuristic allows us to instantly pinpoint the unlimited behaviour of $\mathcal{B}(1)$, circumventing the hefty price of computing the stabilisation monoid.

5.4 The Algorithm

Compute the regular expression e_{LC} of star height $\mathrm{LC}(\mathcal{A})$
Compute a subset automaton, and minimise it
$k = 0$
Repeat
    Construct $\mathcal{B}(k)$
    Check whether the $\sharp$-expressions induced by e_{LC} are witnesses of $\mathcal{B}(k)$
        If an unlimited witness is found, increment k
        Otherwise, check $\mathcal{B}(k)$ for limitedness
            If $\mathcal{B}(k)$ is not limited, increment k
Until $k = \mathrm{LC}(\mathcal{A})$ or $\mathcal{B}(k)$ is limited
Return k

One may wonder whether the two optimisations, namely the Loop Complexity heuristic and minimising the subset automaton, indeed provide a speed-up. As an evidence that it does, we report on the following experiment. We enumerated 200 automata with three states and computed their star height. The computation was considered an overflow when the number of pending matrix products was >5 billions.

Settings	**# Overflows**	**AvgTime(s)**	**Avg monoid dim - size**
No optimisation	5	12.5	62.1 - 1328.0
Loop Complexity heuristic (LCH)	4	10.2	
Minimization	0	6.4	45.7 - 489.1
Minimisation + LCH	0	3.5	

Conclusions

After more than 50 years of research, the star height problem still had the reputation to be intractable, even for very small automata. By implementing state-of-the-art algorithms together with new heuristics and optimisations, we reached a new step in the understanding of this problem. In particular, we discovered a relationship between expressions of optimal loop complexity and unlimited witnesses, which could be of theoretical interest. Our tool Stamina shows that one can compute the star height in non-trivial cases, as it has been successfully tested on several examples of different

nature. It is also a drastic improvement on its previous version ACME for computing limitedness of B-automata and value 1 for leaktight probabilistic automata.

Acknowledgments. We thank Thomas Colcombet for explaining to us the minimisation algorithm for subset automata.

References

[CKLB13] Thomas Colcombet, Denis Kuperberg, Christof Löding, and Michael Vanden Boom. Deciding the weak definability of Büchi definable tree languages. In *CSL*, pages 215–230, 2013.

[CL08a] Thomas Colcombet and Christof Löding. The Nesting-Depth of Disjunctive μ-Calculus for Tree Languages and the Limitedness Problem. In *CSL*, pages 416–430, 2008.

[CL08b] Thomas Colcombet and Christof Löding. The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata. In *ICALP*, pages 398–409, 2008.

[Coh70] Rina S. Cohen. Star Height of Certain Families of Regular Events. *Journal of Computer and System Sciences*, 4(3):281–297, 1970.

[Col09] Thomas Colcombet. The Theory of Stabilisation Monoids and Regular Cost Functions. In *ICALP*, pages 139–150, 2009.

[Col13] Thomas Colcombet. Regular Cost-Functions, Part I: Logic and Algebra over Words. *Logical Methods in Computer Science*, 9(3), 2013.

[Egg63] L. C. Eggan. Transition Graphs and the Star-Height of Regular Events. *The Michigan Mathematical Journal*, 10:385–397, 1963.

[FGKO15] Nathanaël Fijalkow, Hugo Gimbert, Edon Kelmendi, and Youssouf Oualhadj. Deciding the value 1 Problem for Probabilistic Leaktight Automata. *Logical Methods in Computer Science*, 11(1), 2015.

[FGO12] Nathanaël Fijalkow, Hugo Gimbert, and Youssouf Oualhadj. Deciding the Value 1 Problem for Probabilistic Leaktight Automata. In *LICS*, pages 295–304, 2012.

[FK14] Nathanaël Fijalkow and Denis Kuperberg. ACME: Automata with Counters, Monoids and Equivalence. In *ATVA*, pages 163–167, 2014.

[GT01] Gösta Grahne and Alex Thomo. Approximate Reasoning in Semistructured Data. In *KRDB*, 2001.

[Has88] Kosaburo Hashiguchi. Algorithms for Determining Relative Star Height and Star Height. *Information and Computation*, 78(2):124–169, 1988.

[IK93] Karel Culik II and Jarkko Kari. Image Compression using Weighted Finite Automata. *Computers & Graphics*, 17(3):305–313, 1993.

[Kir05] Daniel Kirsten. Distance desert automata and the star height problem. *Theoretical Informatics and Applications*, 39(3):455–509, 2005.

[KMT04] Frank Katritzke, Wolfgang Merzenich, and Michael Thomas. Enhancements of Partitioning Techniques for Image Compression using Weighted Finite Automata. *Theoretical Computer Science*, 313(1):133–144, 2004.

[LS02] Sylvain Lombardy and Jacques Sakarovitch. Star Height of Reversible Languages and Universal Automata. In *LATIN*, pages 76–90, 2002.

[Moh97] Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2):269–311, 1997.

[Rab63] Michael O. Rabin. Probabilistic Automata. *Information and Control*, 6(3):230–245, 1963.

[Sim94] Imre Simon. On Semigroups of Matrices over the Tropical Semiring. *Theoretical Informatics and Applications*, 28(3-4):277–294, 1994.